

Python! Line to Buffered 3D points

Contributed by Kevin Bell
 13, Jan. 2011
 Last Updated 13, Jan. 2011

Quite a while ago our planners needed height data for buildings, so I wrote up a tool that allows them to draw a line (east/west or north/south) and provide a buffer distance, and it will return a point shapefile for the LiDAR strikes within that extent. Preprocessing all of the data before hand might be a more simple solution, but they didn't need the whole city, just a few areas at a time. One twist is that the line might cross a few different raw tiles, so you have to handle them individually. Watch out for the hardcoded paths. You need the LiDAR index and the raw LiDAR xyz files to run this, and it's not too fast considering how much data it has to read.

I haven't seen any code posted recently, so I figured I'd throw something out for your entertainment.

```
# return3DpointsFromInputLineAndBuffer.py
# author: Kevin Bell
# organization: Salt Lake City Transportation
# date: 20090616
# purpose: return a 3D point shapefile of raw LiDAR
#           xyzi data within the extents of the buffered input line.

#ArcGIS Toolbox Parameters:
# Name:      DataType   Type   Direction   default   Schema
# input line FeatureSet  Required Input          \\\rostam\\lidar\\rawxyz\\lineTemplate.shp
# width      String     Required Input          5
# output     FeatureClass Required Output

import sys, os, csv, arcgisscripting

gp = arcgisscripting.create(9.3)
gp.workspace = r"F:\gis\zOtherPeople\20090603_planning_LiDARprofiles"
gp.overwriteoutput = 1
gp.addmessage("beginning Kev's script")

clipper = sys.argv[1]
if clipper == '#':
    clipper = "in_memory\\{8B57910F-BFB0-4E32-BC2C-CCEC1D22BB38}" # provide a default value if unspecified

bufDist = sys.argv[2]

gp.Buffer_analysis(clipper, "in_memory\\bufferedLine", bufDist)
gp.addmessage("buffer complete")
gp.MakeFeatureLayer_management (r"\rostam\lidar\rawxyz\index.shp", "fc_indexLYR")
gp.addmessage("made index layer")
gp.MakeFeatureLayer_management ("in_memory\\bufferedLine", "lineBuffLYR")
gp.addmessage("made buffer layer")
gp.SelectLayerByLocation_management("fc_indexLYR", "INTERSECT" , "lineBuffLYR")
gp.addmessage("selected intersecting LiDAR tiles")

dsc = gp.describe("lineBuffLYR")
ext = dsc.extent
gp.addmessage("described line buffer extents")
strExt = str(ext)
cleanExt = strExt[:strExt.find("NaN")]
cleanExt = cleanExt.strip()
xleft, ybottom, xright, ytop = cleanExt.split(" ") ## have extents here
#print xleft, xright
#print ybottom, ytop
xleft = float(xleft)
xright = float(xright)
ybottom = float(ybottom)
```

```

ytop = float(ytop)

xyzFileList = []
rows = gp.searchcursor("fc_indexLYR")
row = rows.Next()
while row:
    xyzFileList.append(row.USNG)
    row = rows.Next()

print xyzFileList ## have files to query here
gp.addmessage(str(xyzFileList))
gp.addmessage("build query file list")

#ooooooooooooooooooooooooooooooo
#ooooooooooooooooooooooo
#ooooooooooooooooooooooo

pointCloudSubset = []

def convertCSVtoPoints(infile):
    infile = r"\rostam\lidar\rawxyz" + "\\" + infile + ".xyz"
    theReader = csv.reader(open(infile))

    print "beginning to read file"
    i = 0
    for r in theReader:
        i += 1
        if str(i).endswith("000000"):
            print i
            gp.addmessage(str(i) + " points queried")
        x = r[0]
        y = r[1]
        z = r[2]
        intensity = r[3]
        # HERE WE NEED TO TEST AGAINST THE EXTENT, AND HOLD IN MEMORY
        if (float(x) >= xleft) and (float(x) <= xright):
            if (float(y) >= ybottom) and (float(y) <= ytop):
                myTuple = (x,y,z,intensity)
                gp.addmessage(str(myTuple))
                pointCloudSubset.append(myTuple)
    #infile.close() #new
    del theReader #new

#ooooooooooooooooooooooo

for f in xyzFileList:
    convertCSVtoPoints(f)
    gp.addmessage("beginning to convert csv to point list")

apath = sys.argv[3]
folder = os.path.dirname(apath)
fc = os.path.basename(apath)
outProj = r"Coordinate Systems\Projected Coordinate Systems\UTM\NAD 1983\NAD 1983 UTM Zone 12N.prj"
gp.CreateFeatureclass(folder, fc, "Point", "", "Disabled", "Disabled", outProj)
gp.AddField(sys.argv[3], "XCoor", "DOUBLE", "16", "3", "", "NON_NULLABLE", "NON_REQUIRED", "", )
gp.AddField(sys.argv[3], "YCoor", "DOUBLE", "16", "3", "", "NON_NULLABLE", "NON_REQUIRED", "", )
gp.AddField(sys.argv[3], "Elev", "DOUBLE", "8", "3", "", "NON_NULLABLE", "NON_REQUIRED", "", )
gp.AddField(sys.argv[3], "Intensity", "DOUBLE", "8", "1", "", "NON_NULLABLE", "NON_REQUIRED", "", )
gp.addmessage(".....created output 3D point file")

pntListLen = str(len(pointCloudSubset))
gp.addmessage(pntListLen + " points for output shapefile")

```

```
cur = gp.InsertCursor(sys.argv[3])
pnt = gp.CreateObject("Point")
featid = 1

for t in pointCloudSubset:
    pnt.x = t[0]
    pnt.y = t[1]
    pnt.z = t[2] ## * 3.2808399 to convert to feet
    Inten = t[3]
    feat = cur.NewRow()
    feat.shape = pnt
    feat.XCoor = pnt.x
    feat.YCoor = pnt.y
    feat.Elev = pnt.z * 3.2808399
    feat.Intensity = Inten
    feat.id = featid
    #feat.fid = featid
    cur.InsertRow(feat)
    featid = featid + 1

del cur #new

print "-----FINISHED-----"
gp.addmessage("finished loading 3D point file")
del gp #new
```